PREVAJANJE PROGRAMSKIH JEZIKOV – analiza in načrtovanje jezika

DEFINICIJA KONSTRUKTOV ZA OPIS MESTA

V nadaljevanju so predstavljene definicije konstruktov našega domensko specifičnega jezika, ki ga bomo izdelali v sklopu Projektnega praktikuma. Jezik bo služil za izris mesta Maribor, pri čemer se naša skupina osredotoča na vse, kar je v Mariboru povezano s kolesarstvom.

Torej prej omenjeni konstrukti, bodo služili za predstavitev kolesarskih entitet, na katere se osredotočamo. Poleg konstruktov bomo dodali tudi ukaze in nekaj drugih znanih konceptov programskih jezikov, za lažji in podrobnejši opis naših entitet.

KAZALO VSEBINE

Avtorji: Marko Roškar, Jakob Oprešnik, Erik Lašič

Mentor: Prof. dr. Milan Zorman, univ. dipl. inž. rač. in informatike

1. ŠTEVILA

Celoštevilske vrednosti (int) ali številske vrednosti s plavajočo vejico (float).

```
10
10.25
```

2. NIZI

```
NIZ
```

3. TOČKE

S točkami predstavimo lokacije na zemljevidu, pri čemer je komponenta X geografska dolžina (angl. longitude), komponenta Y pa geografska širina (angl. latitude). S točkami bomo omejili bloke s katerimi bomo opisali entitete v mestu. Koordinata točke je lahko pozitivno ali negativno predznačena številska vrednost ali spremenljivka ali konstanta ali pa izraz.

```
(X, Y)
```

Lokacije, katere so za nas pomembne je zato smiselno predstaviti s točkami, na naslednji način. Te lokacije so kolesarska stojala (bike stand), kolesarnice (bike shed) in izposojna mesta Mbajk.

```
bikeStand(INT) NAME { POINT }

bikeShed NAME { POINT }

mbajk NAME { POINT }
```

4. BLOKI

Bloki so konstrukti, s katerimi bomo opisali entitete v mestu, kot so zgradbe, ceste, reke, parki ipd., kot tudi samo mesto.

```
city NAME {
  BLOCKS
}
```

Blok road predstavlja cesto, ki bo vseboval ukaze za izris odseka ceste (oz. ulice).

```
road NAME {
  COMMANDS
}
```

Blok `bikePath` predstavlja kolesarsko stezo, ki bo vseboval ukaze za izris odseka kolesarske steze.

```
bikePath NAME {
  COMMANDS
}
```

Blok `bikeTourPath` predstavlja kolesarsko turistično pot. Te se od kolesarskih stez razlikujejo v tem, da so načeloma povezane s kolesarskimi potmi, ki segajo daleč izven okvirov mesta, tudi v ostale bližnje občine.

```
bikeTourPath NAME {
  COMMANDS
}
```

Blok `bikeCoridor` predstavlja kolesarski koridor. Kolesarski koridorji služijo kot nekakšna hrbtenica kolesarskega omrežja znotraj mesta.

```
bikeCoridor NAME {
  COMMANDS
}
```

Blok `building` predstavlja zgradbo, vseboval bo ukaze za izris obrobe zgradbe, pri čemer mora biti izris obrobe zaključen lik.

```
building NAME {
  COMMANDS
}
```

Blok `river` predstavlja reko. Podobno kot blok road bo vseboval ukaze za izris reke.

```
river NAME {
  COMMANDS
}
```

Blok `park` predstavlja park oz. zeleno površino znotraj mesta. Ta bo, podobno kot blok building vseboval ukaze za izris njegove obrobe, pri čemer mora biti izris obrobe tudi tu zaključen lik.

```
park NAME {
  COMMANDS
}
```

5. UKAZI

V prejšnjem podpoglavju opisani bloki vsebujejo naslednje ukaze.

Ukaz `line` preprosto izriše črto, med dvema podanima lokacijama. Tretji argument predstavlja širino ceste.

```
line(POINT, POINT, WIDTH)
```

Ukaz bend izriše krivuljo, med dvema podanima lokacijama, kot tretji argument pa podamo vrednost ukrivljenosti (v stopinjah °). 0° bo pomenilo izris ravne črte, 45° bo pomenil izris približno četrtine krožnice, 90° pa še ostrejši kot itd. Pozitivni koti pomenijo izris v levo nagnjene krivulje, medtem ko negativni koti pomenijo izris v desno nagnjene krivulje.

```
bend(POINT, POINT, ANGLE)
```

Ukaz box izriše pravokotnik, med podanima lokacijama, pri čemer prva lokacija pomeni zgornji levi kot pravokotnika, druga lokacija pa spodnji desni kot pravokotnika.

```
box(POINT, POINT)
```

6. SPREMENLJIVKE IN KONSTANTE

Konstrukt `var` omogoča, da ustvarimo novo spremenljivko, v kateri bomo hranili neko številsko vrednost, vrednost neke druge spremenljivke, točko ali pa vrednost, ki jo vrne funkcija. To zna biti koristno, saj nam bo omogočalo ponovno uporabo vrednosti, pa tudi naknadno spreminjanje te vrednosti. Ob koncu deklaracije spremenljivke ali konstante je obvezno podpičje.

```
var ime = ŠTEVILO
```

Konstrukt `const` omogoča, da ustvarimo novo konstanto, v kateri lahko hranimo enak nabor vrednosti kot pri spremenljivki, vendar te vrednosti ne bo možno naknadno spreminjati.

```
const ime = ŠTEVILO
const ime = POINT
```

Za podporo deklariranja spremenljivk in konstant moramo seveda dodati tudi prireditveni operator = .

7. IZRAZI

Izrazi so sestavljeni iz spremenljivk, konstant številskega tipa in aritmetičnih operatorjev med njimi. Operatorji so +, -, *, /. Izraze lahko tako kot število podamo kot koordinato točke, ali kot katerikoli drugi parameter pri ukazih `line`, `bend`, `box` in `circ`, lahko pa jih tudi priredimo neki novi spremenljivki.

```
var name1 = 5
var name2 = name1 + 10
var name3 = name2 – 3
```

8. ZANKE IN VEJITVE

Za lažji izpis ponavljajočih se elementov nam lahko prav pride zanka `for`. Dodatno pomoč pri tem pa nam lahko zagotovi tudi vejitev `if else`.

Zanko začnemo z rezervirano besedo `for`, kateri sledi začetna spremenljivka ali število, zatem sledi rezervirana beseda `to`, za njo pa še končna spremenljivka ali število. Ukaze, ki se bodo izvedli znotraj vsake iteracije pa navedemo znotraj zavitih oklepajev in sicer vse v isti vrstici.

```
for (variable to variable) { COMMANDS }
```

Vejitev oz. pogojni stavek začnemo z rezervirano besedo `if`, kateri nato sledi dvopičje, zatem spremenljivka ali število, primerjalni operator in spet spremenljivka ali število. Blok vejitve, tako kot pri zanki, označimo z zavitimi oklepaji in znotraj navedemo ukaze. Za blokom `if` lahko opcijsko sledi eden ali več `elseif` blokov, ki prav tako kot blok `if` vsebujejo pogoj in pred njim dvopičje, celotna vejitev pa se vedno konča z `else` blokom, kateremu tudi obvezno sledi dvopičje, vendar tu ne navajamo pogoja.

```
if  (variable OPERATOR variable) { COMMANDS }
elseif (variable OPERATOR variable) { COMMANDS }
else { COMMANDS }
```

Primerjalni operator je eden izmed naslednjih šestih operatorjev: <, <=, >, =>, ==, !=

9. ABSTRAKCIJE

Če želimo določene kompleksnejše elemente večkrat ponoviti na poljubnih mestih, nam pri tem lahko delo zelo olajšajo funkcije oz. procedure. V funkcijo podamo enega ali več parametrov in na podlagi njunih vrednosti funkcija vrne novo vrednost.

Definicija funkcije se začne z rezervirano besedo `func`, kateri sledi ime funkcije, ki ga določimo sami, zatem pa znotraj oklepajev sledijo funkcijski parametri. Parametri so lahko ali spremenljivka ali pa število. Z rezervirano besedo `return` zaključimo izvajanje programa znotraj funkcije. Funkcija lahko vrača blok, točko, število ali spremenljivko s številsko vrednostjo.

```
func name(args) {
  COMMANDS
  return POINT
};
```

Klic funkcije se začne z uporabo rezervirane besede `call`, kateri sledi ime funkcije, znotraj oklepajov pa podamo parametre. Parametri so ločeni z vejicami.

```
call name(callargs)
```